AD A091766

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

DISTRIBUTED SENSOR NETWORKS

SEMIANNUAL TECHNICAL SUMMARY REPORT
TO THE
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

1 OCTOBER 1979 – 31 MARCH 1980

ISSUED 25 SEPTEMBER 1980

Approved for public release; distribution unlimited.

LEXINGTON MASSACHUSETTS

ABSTRACT

Distributed Sensor Network research in the areas of tracking, signal processing, and system software is reported as is progress with the design and development of a DSN test bed. Two-node acoustic location algorithms have been reformulated to better fit into the real-time DSN environment. Deghosting techniques have been investigated and the general organization of tracking tasks further refined. The development of hardware and software for a small acoustic array and data acquisition system which will evolve into a node of a DSN test bed has been completed and the data acquisition system is operational. System software designs and ideas for initial three-node test bed use and for more general DSN systems are presented.

Accession For
NTIS [illegible]
DTIC [illegible]
[illegible]
[illegible]
[illegible] Codes
[illegible]
Special
A

CONTENTS

CONTRIBUTORS TO DISTRIBUTED SENSOR NETWORK PROGRAM DURING THIS REPORTING PERIOD

GROUP 22

Lacoss, R. T.
Eskeli, D. R.
Green, P. E.
Landers, T. E.
Retzlaff, A. T.
Rickless, R. S.
Walton, R. L.

DISTRIBUTED SENSOR NETWORKS

I. INTRODUCTION AND SUMMARY

This Semiannual Technical Summary (SATS) for the Distributed Sensor Networks (DSN) program reports research results for the period 1 October 1979 through 31 March 1980. The DSN program is aimed at developing and extending target surveillance and tracking technology in systems that employ multiple spatially distributed sensors and processing resources. Such a DSN would be made up of sensors, data bases, and processors distributed throughout an area and interconnected by an appropriate digital data communication system. It would serve users who are also distributed within the area and serviced by the same communication system. The case of particular interest is when individual sensors cannot view the entire surveillance area and when they can individually generate only limited information about targets in their field of view. The working hypothesis of the DSN program is that through suitable netting and distributed processing the information from many such sensors can be combined to yield effective and serviceable surveillance systems. Surveillance and tracking of low-flying aircraft, including cruise missiles, using sensors that individually have limited capabilities and limited fields of view, has been selected to develop and evaluate DSN concepts in the light of a specific system problem. The research plan is to investigate these concepts and to develop a DSN test bed which will make use of multiple small acoustic arrays to detect and track low-flying aircraft.

Research in the area of multiple-node acoustic tracking is reported in Sections II and III below.

Section II presents the overall structure of the multinode tracking task and the details of a new two-node location algorithm which is planned to be the basis for initial on-line experiments with real multinode acoustic azimuth measurements. The new two-node target location formulation is convenient for real-time use. Each node can always make immediate use of its own most recent azimuth measurements, in conjunction with azimuth measurements from neighbors, to produce the most up-to-date target location. Two-node, single-target simulation results are presented and demonstrate the operation of the algorithm. The existence of false targets as well as true target locations is noted with some suggestions for how to identify false targets.

Multiple targets in a multiple-node environment introduces additional false target problems. If azimuth measurements from two different nodes are combined to produce locations but the measurements do not actually correspond to the same target, then a false target called a "ghost" will result. Section III reports the results of a study of deghosting techniques for azimuth-only sensors. Simulations were used to demonstrate various deghosting aids such as recognition of unreasonable track dynamics, multiple-node redundancy, and consistency requirements for estimated source power levels. These and other aids will apply in a DSN because sensor ranges tend to be on the order of node separations and also targets are within range of different sensors at different times.

Section IV reports progress with our single-node acoustic data acquisition system which is now operational. The hardware is described with particular attention to measures taken to achieve low system noise, equivalent to 20-dB sound level, which is below ambient except in specially designed acoustic chambers. The gain-ranged analog-to-digital conversion system operates from that floor to well over 100 dB, which is sufficient to deal with even very large aircraft at very close ranges. The status of the data acquisition software is also summarized. The software kernel (DAK) which supports data acquisition is to be modified and used for multi-node experiments involving signal processing as well as data acquisition. The modifications, involving system calls to access more memory, are described. The first three test-bed DSN nodes to be deployed will initially use this modified software and the hardware will be like the data acquisition system, with signal-processing hardware added.

Work in two signal-processing areas is reported in Sec. V. An interactive data analysis package has been developed to process data produced by the data acquisition system, and we have started work on the software to make use of array processors for real-time DSN signal processing. The interactive package is in use on our PDP-11/70 running the UNIX operating system. Depending on options selected, it will output waveforms, spectra, or azimuthal power files in well-defined formats for which access and manipulation packages are available. Use of this package is just starting. It will be utilized to help select single-node signal-processing parameters. Modules for single-node azimuth tracking and multinode tracking will be added as work on such algorithms proceeds. The second part of Sec. V reviews the hardware planned for signal processing at each DSN test-bed node and outlines plans for providing a user interface for programs running in the host computer included in each node.

Section VI reports on research in the area of system software designed for use in a real-time distributed network such as a DSN. The data acquisition kernel (DAK) which has been developed and is being modified for signal processing as well as data acquisition tasks is adequate for initial DSN experiments. DAK is a single computer system and the DSN must ultimately develop a truly multicomputer system. The multicomputer design ideas presented in Sec. VI are an extension of ideas already incorporated in DAK. In particular, all interprocess communication is accomplished by means of structured queueable objects which are exchanged between processes. Topics discussed include queue organization for a distributed environment, passing of queues or processes between computers, and reliability.

II. ACOUSTIC TRACKING ALGORITHMS

The major technical achievements in the area of acoustic tracking have been the development of a new method for position location, and the subsequent refinement of the tracking task. The new position location method, which is described in the next section, enables target locations to be determined by a pair of nodes in a manner that is easy to incorporate within a Kalman or similar filtering scheme.

Using this method, the levels of tracking are:

(a) Individual nodes form azimuth tracks for targets and broadcast new points, as they are determined, to neighbors.

(b) Nodes use received azimuth track data, combined with their own azimuth data, to determine target locations on a node pairwise basis. These node pair locations are broadcast to neighbors, along with an estimate of the accuracy of the locations.

(c) Nodes use received locations, along with their own locally generated target locations, to determine target spatial tracks. These data are then communicated to users and other nodes to satisfy overall system operational requirements.

In forming single site azimuth tracks, we plan to use a tracking algorithm patterned after the work of Reid* and Kevarian.† Recursive filtering and estimation will be used to smooth data, obtain estimates of the azimuth vs time tracks, and obtain estimates of azimuth variances.

The azimuth variances for in-track targets will be used to form azimuth gates as shown in Fig. II-1. Peaks found to fall within the gates will be associated with that target. Peaks, such as C in Fig. II-1, will be recorded for an appropriate acquisition time (probably 5 sec). If the peak appears at close azimuths in a statistically significant number of plots (probably 80 percent), then a new target track and filter will be initiated provided that the variance is within acceptable limits. This method avoids generating large numbers of new tracks at each time step which would have to be eliminated as being false at a later time.

When the gates for two peaks overlap, as shown in Fig. II-2, we then modify our techniques to allow for the fact that there is a probabilistic association between each peak and both existing tracks. For this situation, both measurements are factored into both filters, based on the closeness of each peak to the predicted track. Also, we will detect when there is no longer any statistically significant difference between the tracks and determine when they should be combined into one track.

When these azimuths are combined into locations of targets, the variances will be carried along, giving an error ellipsoid for each target location as shown in Fig. II-3. These target positions will be formed for the last point of common observation by the pair of nodes. If one node has a more recent observation on a node, these data will not be used until a corresponding observation is received from another node. This means that the locations are for the most recent time in the past that a location can be formed by any pair of nodes.

*D. B. Reid, "An Algorithm for Tracking Multiple Targets," IEEE Trans. Auto. Cont. AC-24, No. 6 (1979).

†K. M. Kevarian, "Multi-Object Tracking by Adaptive Hypothesis Testing," B. Sc. Thesis, Department of Electrical Engineering, Massachusetts Institute of Technology (1979).

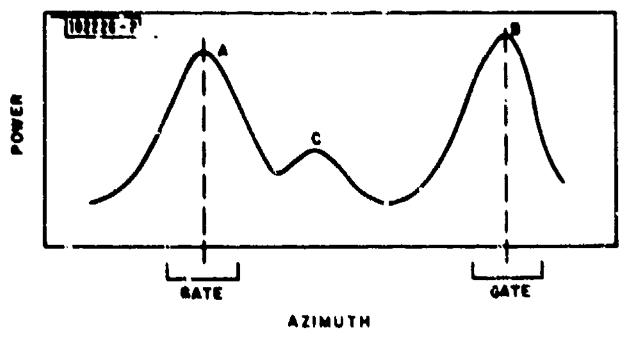

Fig. II-1. Power vs azimuth at a single node. Azimuth gates are shown for resolved targets.

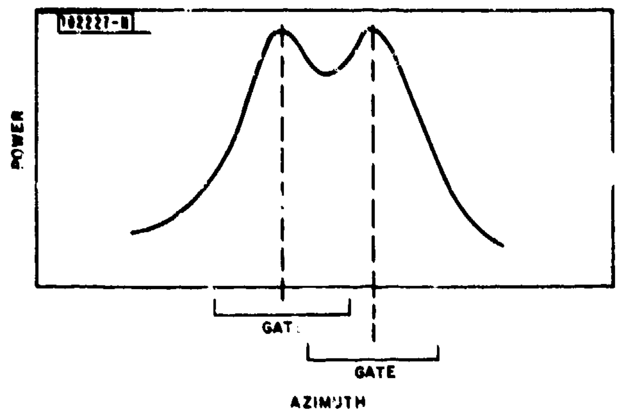

Fig. II-2. Power vs azimuth showing overlapping azimuth gates.

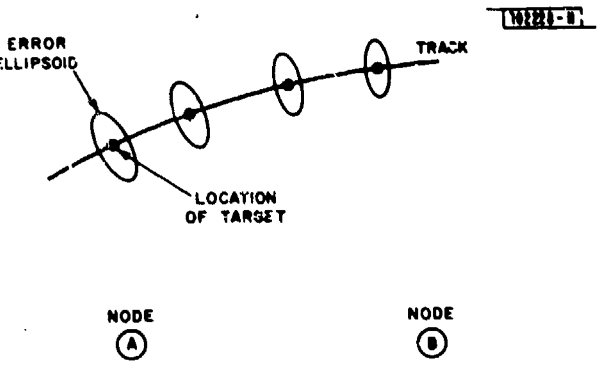

Fig. II-3. Spatial determination of target track using data from two nodes.

These two-node positions can then be formed into a track by any node that generates or receives the position reports. The positions will be formed into tracks using Kalman filters to smooth the data and to get an estimate of the variances in target location and velocity estimates. Two-node trackers may also deal directly with single-node azimuth tracks as well as with the two-node location estimates. In general, segments of track will be formed by pairs of nodes as a target flies through a DSN network. The locations from these multiple segments of track can then be combined into a single track by any node that receives the location estimates. Other multiple site trackers which are driven by azimuth rather than location estimates are also possible and will be investigated in the course of our ongoing work.

Multiple site tracking using location rather than azimuth inputs reduces the spatial tracking to a form that is similar to radar tracking. In doing so, we are hoping to use many of the techniques that have been developed in this field. The major differences are that a track formed from observations by nodes A and B is not statistically independent from a track formed from observations by B and C, and possibilities of target ambiguities or ghosts exist as discussed in Sec. III of this report.

A. TARGET LOCATION ALGORITHM

Our prior method for position location involved finding the intersection of two curves in space. These curves were generated from the azimuth vs time curves, for some time in the past, T, as shown in Fig. II-4 by projecting the azimuths at 1-sec intervals forward from T and equating these to increments of C meters in range (where C is the velocity of sound).

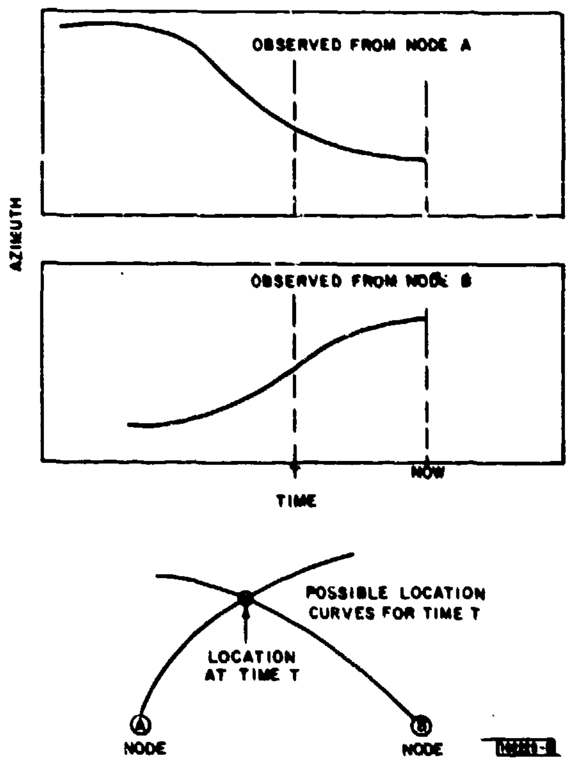

Fig. II-4. Determination of position location using possible position curves.

However, we would like to update the state of the target every time we determine a new azimuth. With our prior method, a particular reading allowed an extra point to be appended to each of the curves for each time, T, in the past. This was cumbersome to implement in a real-time environment because of such factors as the need to maintain possible position curves for all times in the past. To overcome real-time implementation problems an alternate method has been developed which is called the "Reflected Observation" method. The following is a description of this method.

Consider the situation where we have a target flying past node A which is observing the target and sending the azimuth vs time data to node B. Node B then makes its first observation on the target at angle ϕ_B. This observation may correspond to an observation made by A, as shown in Fig. II-5. If the travel time of sound from the target to A is t_A and from the target to B is t_B, then A would have observed the target at a time $\delta_t = t_B - t_A$ in the past. As shown in Sec. C below, we can relate δ_t to the angles and the distance d between the nodes by

$$\delta_t = \frac{d}{C} \frac{[\sin(\phi_A) - \sin(\phi_B)]}{\sin(\phi_A + \phi_B)} \quad .$$

This equation is only valid for the range in which the angular observations from A and B intersect.

Using this formula, we can translate the observation ϕ_B to a curve of possible observations of δ_t vs ϕ_A as shown in Fig. II-6. The intersection of this reflection curve with the curve of

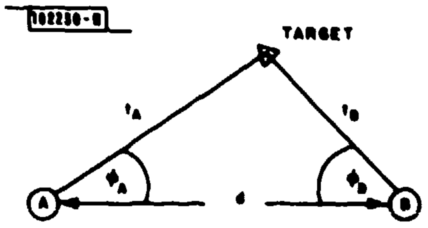

Fig. II-5. Target observed by A and B.

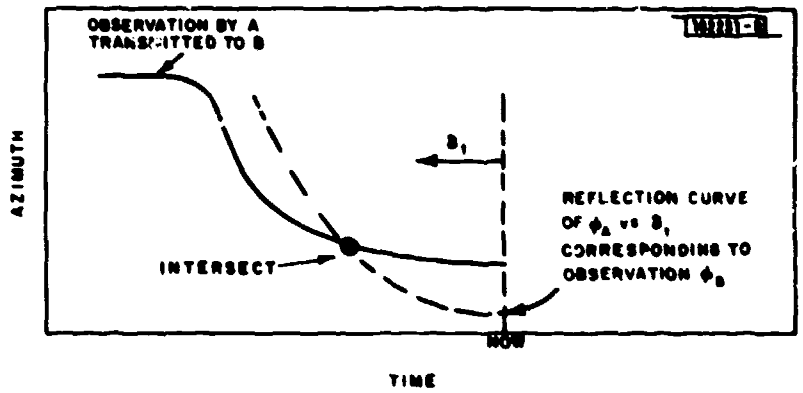

Fig. II-6. Intersection of actual observations by A with reflection to node A observation by B.

observations from A gives the angle of observation of A that is common with the angle of observation of B for which the reflection curve is generated. The common observation azimuths can then be triangulated to find the location of the target. The time at which the target was at this location can then be determined by subtracting the travel time from the location to node B from the time of observation at node B. This position location can then be used as input to a conventional Kalman spatial filter tracking scheme, except that the filter runs at a time that corresponds to the last time which gave rise to a common observation at nodes A and B. As B makes each new observation, it can compute a new position, and use this as its input to the Kalman filter.

There are several special cases of interest. The first is when the reflected curve does not intersect the observed curve. In this case, the observation at B does not correspond to a prior observation by A. B could save up its results and wait for A to make an observation. It is probably better for B to just transmit the observation to his neighbors. In this way, the node making the confirming observation will do the processing. At first look, this would seem to distribute the processing load in the network in an effective manner. The procedure would be for a node to always transmit its observation, but only to use its current observation in forming and updating tracks.

The second case is where there are multiple observation curves from A as shown in Fig. II-7. In this case, there will be ambiguities which will need additional discriminants. We can use spectra or an observation from a third site C. If we reflect ϕ_B onto the observations from C, we should find only one intersect corresponding to the intersects on the observations from A. If there is more than one, then we must assume that there is more than one target and track it accordingly. In general, the problem of multiple observation curves is a major topic of continuing research. Incorrect associations result in "ghost" targets which do not occur and which must ultimately be identified and removed (see Sec. III).

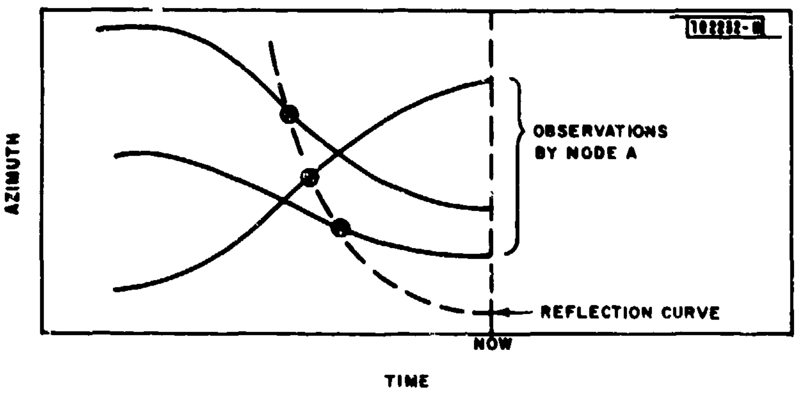

Fig. II-7. Multiple observation curves from A showing possible intersects with reflection curve.

We also find that the reflection curve can intersect the observation curve at more than one place. This implies ambiguity in the position location. It may be possible to eliminate one or the other of the ambiguous locations by comparing it with the current target state. Otherwise, it will be necessary to track both targets until one or the other is proved to be the true target.

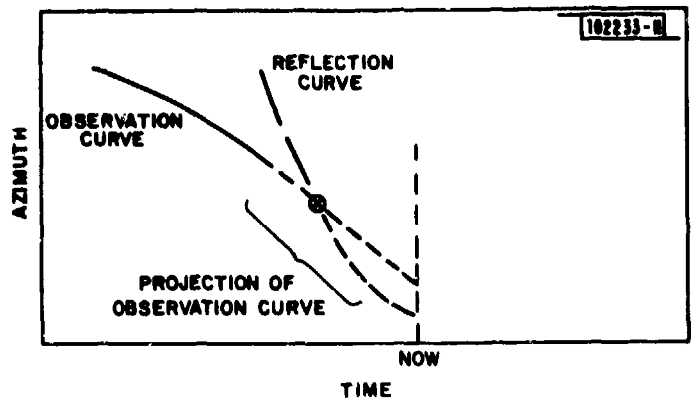

Fig. II-8. Use of projected observation curve for target location.

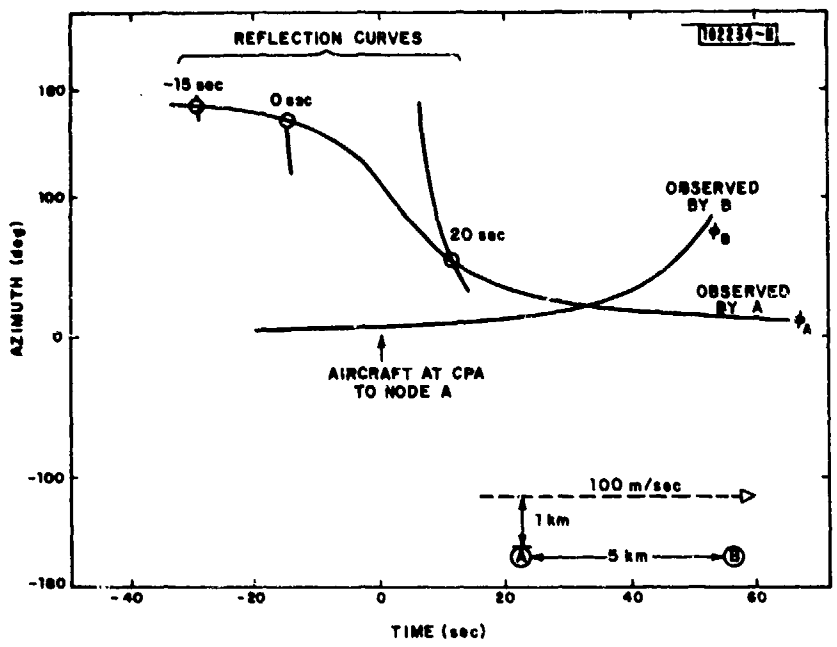

Fig. II-9. Observations and reflections for aircraft flying past two nodes.

There is also the case where A was observing the target, but lost it before B acquired it as shown in Fig. II-8. In this case, it may be possible to project the observation curve to intersect with the reflection curve, to obtain a location for the target. In the case that A actually had the target in location track in conjunction with another node, that location track could be extrapolated and converted to a projected azimuth observation curve for use with the B observation.

B. SOME RESULTS USING THE REFLECTED OBSERVATION LOCATION METHOD

Figure II-9 shows the simulation of the azimuths at which peaks would be observed by two nodes, A and B, for an aircraft flying parallel to them. The aircraft is assumed to have a closest point of approach (CPA) of 1 km from node A, at time 0, and is flying due east at 100 m/sec. The observations of B at times of −15, 0, 20, and 40 sec relative to the CPA at node A were converted into reflection curves. The intersect of these curves with the curve of observations from node A gives the angles of observation at node A corresponding to the angles of observation from node B. These angles were then triangulated to find the locations at the times of common observation, and these were shown to be on the simulated aircraft track. In Fig. II-9, the reflection curves are plotted for all angles observed at A up to the time of observation at B for which the reflection curve is plotted. For this reason, the curves for later reflection times are longer. The curve for a reflection time of 40 sec no longer intersects with the curve from node A. This is because the aircraft was closer to B than to A, and A had not yet received the sound that B was observing. Strictly speaking, the curve for 40 sec should be terminated at 40 sec since that is the time of observation.

Figure II-10 shows the results obtained with the aircraft flying past the same two nodes, except that the course has been changed so that the aircraft flies between the nodes with its CPA to node A being slightly closer than its CPA to node B. Again reflection curves were

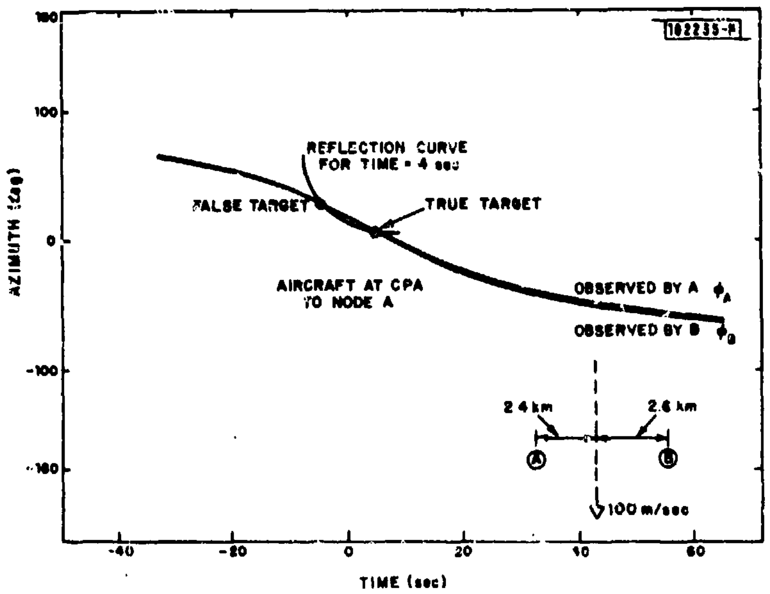

Fig. II-10. Observations and reflections for an aircraft flying between two nodes showing false targets.

generated, and the aircraft was tracked. In this case however, a false target appeared in the form of a second intersection between the reflection curve and the observations by A. In Fig. II-10 the reflection curve for only one time is shown, which is during the time that false targets were observed. The geometrical explanation of the false target can be seen from Fig. II-11. In determining the intersect, we are only using a singular observation of B. There are two observations that A made prior to B for which the time difference of arrival (TDOA) fits the mutual observations of A and B. As a consequence, we have no way of determining from a singular measurement made by B which is the true target.

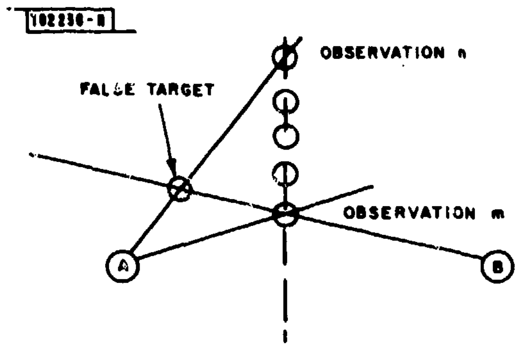

Fig. II-11. Geometrical explanation for the false targets shown in Fig. II-10.

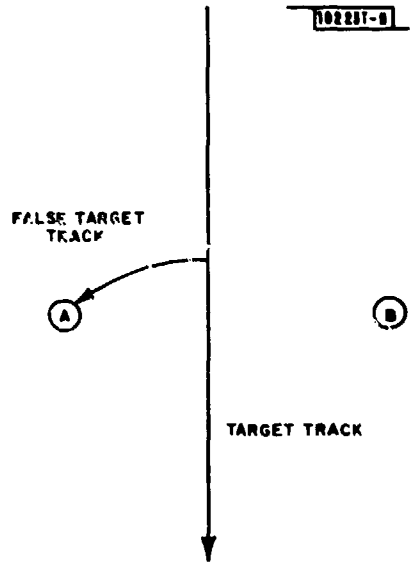

Fig. II-12. Tracks for target and false target corresponding to Fig. II-10.

Figure II-12 shows the track for the true and false targets. It can be seen that the false target appears in the vicinity of the nodes and "flies" from the true track into one of the nodes. This false target can be eliminated using several possible mechanisms. First, if B had been able to make a prior observation corresponding to A's observation n (see Fig. II-11), then he would be able to tag that observation as already having been used. This point would then not be used in generating the reflection curve and the false track would be eliminated. Alternately, if an intervening hill had precluded B from making this prior observation, then it would be necessary to track the false target until it disappeared by flying into node A. As both the true and false tracks are supported by B's observation, it may well be possible to eliminate the false target on the basis that it is competing against a well-established track for the use of the singular observation. Thus, using decision making techniques, we may well be able to eliminate such false targets long before they would be displayed to operators.

C. DERIVATION OF REFLECTION CURVE

Following is the derivation of the reflection curve used for the reflected observation location method presented and discussed above.

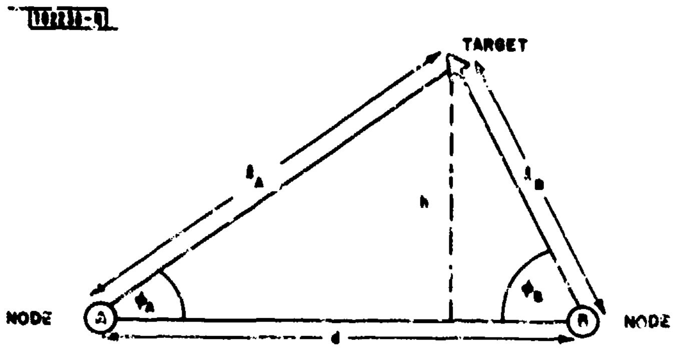

Fig. II-13. Geometry for determining reflection curve.

Consider the geometry shown in Fig. II-13. For that geometry,

$$\ell_A = \frac{h}{\sin\phi_A} \quad , \quad \ell_B = \frac{h}{\sin\phi_B} \quad , \tag{1}$$

$$\delta_t = \frac{\ell_B}{C} - \frac{\ell_A}{C} \quad , \tag{2}$$

$$\ell_A \cos\phi_A + \ell_B \cos\phi_B = d \quad , \tag{3}$$

and from Eq. (1),

$$\ell_B = \ell_A \frac{\sin\phi_A}{\sin\phi_B} \quad . \tag{4}$$

Substituting into Eq. (3) gives

$$\ell_A \cos\phi_A + \ell_A \frac{\sin\phi_A}{\sin\phi_B} \cos\phi_B = d \quad ,$$

$$\ell_A = \frac{d \sin\phi_B}{\sin\phi_B \cos\phi_A + \sin\phi_A \cos\phi_B} \quad ,$$

$$\ell_A = \frac{d \sin\phi_B}{\sin(\phi_B + \phi_A)} \quad ,$$

$$\ell_B = \frac{d \sin\phi_A}{\sin(\phi_B + \phi_A)} \quad ,$$

and finally

$$\delta_t = \frac{d}{C} \frac{(\sin\phi_A - \sin\phi_B)}{\sin(\phi_B + \phi_A)} \quad .$$

This is valid only in the range where the observations intersect, namely:

if $\phi_B > 0$: $0 < \phi_A < \pi - \phi_B$

if $\phi_B < 0$: $0 > \phi_A > -\pi - \phi_B$

if $\phi_B = 0$: then $\phi_A = n\pi$ for $n = 0, 1, 2, 3 \ldots$

and the position must be determined by the intersection of the ray from node B with a projection of the prior observations.

III. DEGHOSTING TECHNIQUES FOR AZIMUTH-ONLY SENSOR NETWORKS

A basic element of target information available from a node using a small array of microphones as sensors is azimuth as a function of time. For a single target, its location is fixed by projecting the azimuth data at two separate nodes to an intersection as discussed in Sec. II above. But beyond the difficulty involved in processing the array data into an azimuth and acoustically projecting that azimuth out in space and back in time is the problem of an intersection that results from azimuths that in reality correspond to different targets (i.e., the problem of ghosting). The problem can be severe since at each pair of sensor nodes n-targets can produce n^2 intersections ($n^2 - n$ ghosts) and in m nodes there are $m(m-1)/2$ pairs. For example, 4 targets simultaneously being tracked by 3 nodes can have 36 ghosts. When targets are at ranges large compared to the node separation, many ghosts can have track characteristics like those of true targets.* A study of the azimuth-only multisensor ghosting problem has been completed and the conclusion reached is that for a DSN there are sufficient deghosting aids available to solve the ghosting problem. For simplicity, the finite velocity of sound has been ignored for this study.

In a DSN, node separation and target detection range are generally comparable distances. Additionally, the field of view of a DSN is largely internal to its perimeter. The effects are that a target of interest is in the field of view of at most a few nodes at any given instant of time and, in general, a target will subtend a large range of angles during the total encounter period it has with any node pair. Since ghosting is strictly a sensor pair phenomenon, it is then reasonable to expect that track dynamics will vary enough to distinguish true targets from ghosts and, since any target will continuously pass from one sensor pair to another, that only true target tracks will be continuous. To quantify this behavior, an m-target n-sensor simulation program has been written and used to investigate the ghosting problem in a number of realistic multitarget scenarios in a DSN geometry.

In addition to purely geometrical considerations, targets are characterized by their sound signatures. Since a ghost is a target pair phenomenon, the acoustic signature attached to a ghost will be inconsistent from node to node. The most basic signature characteristic is signal power. The targets producing sound may be much closer or farther from the sensor nodes than the ghosts. If so, the sound power apparently being emitted by a ghost may be larger or smaller than is reasonable for an aircraft. In most cases, real targets will be at different ranges from sensing nodes so that the power attributed to the ghost by each of the two detecting nodes will differ. When a ghost results from detecting two targets with different propulsion systems an even stronger discriminant will exist. For example, a ghost due to an azimuth on a helicopter and one on a jet will have a widely different spectra at the two detecting nodes. This spectral property is very robust and nearly independent of target range and aspect. Its proper application to deghosting should occur before tracking is initiated so that location algorithms will be run only on data with consistent target signatures. For this reason, the simulator does not consider target spectra. However, the apparent power deghosting characteristic is included.

Two other deghosting aids have also been considered: redundancy and the consistency of the appearance and disappearance of target tracks. Redundancy here implies the number of confirming nodes contributing to track. It is, in the geographically local sense, a rather weak

* W. P. Harris, C. D. Forgie, and F. C. Frick, "The JAMTRAC Study," Technical Report 180, Lincoln Laboratory, M.I.T. (8 July 1958), DDC AD-133860.

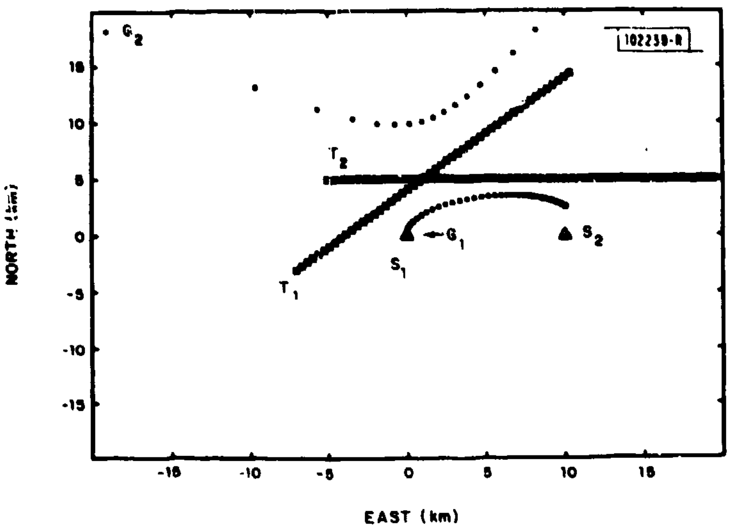

Fig.III-1. Ghost tracks due to 2 targets (T_1, T_2) observed by 2 azimuth-only sensor nodes (S_1, S_2). Targets are progressing eastward. Successive locations are plotted at constant 2-sec intervals. Ghost G_1 is due to the false intersection of S_1 strobed on T_2 and S_2 on T_1. Ghost G_2 is due to S_1 on T_1 and S_2 on T_2. G_1 is born at S_1 and dies near S_2. G_2 is born at very large range to the NW and recedes at very high velocity to the NE.

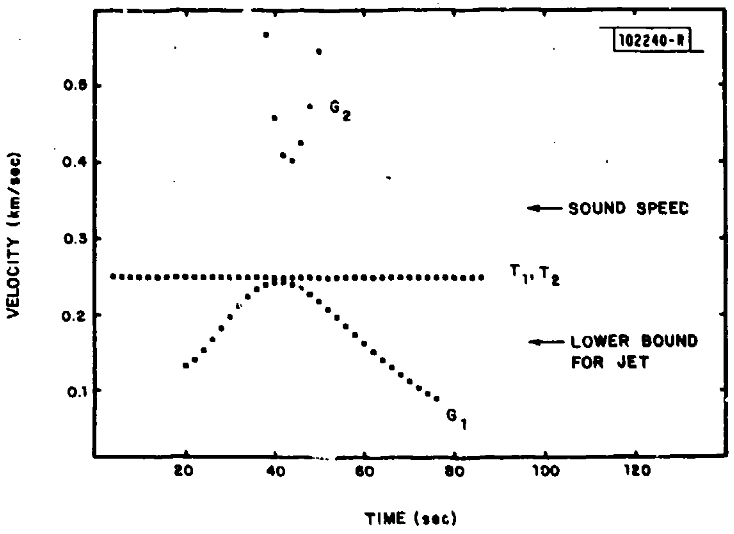

Fig.III-2. Velocity of the ghost tracks in Fig.III-1. G_2 remains supersonic over its entire path. G_1 exhibits a wide range of subsonic velocities.

discriminant but in the network sense probably the best discriminant. A ghost location is a two-target two-sensor effect and the location of a ghost is a function of the specific two-target two-sensor geometry. The likelihood of one sensor pair generating a complete ghost track that will be confirmed by adjacent sensor pairs is small. It is very unlikely that a ghost will form a reasonable track across the network. However, in the local sense where redundancy is measured by the number of simultaneous azimuths contributing to a single location, the situation is not as clear. Even though a ghost has two and only two contributing azimuths it is easy to find realistic examples where ghosts will cluster within the system location resolution. Since the economy of a DSN is maximized when it is constructed such that targets are not simultaneously being detected at very large numbers of nodes, local redundancy then cannot be used exclusively to eliminate ghosts. As will be seen in the simulation examples, ghosts often appear and disappear at inconsistent places; for example, at a node or well beyond any possible sensor detection range. This characteristic then can also be used for deghosting.

The simulator is constructed as follows. Targets are specified by their initial location, acoustic power level, and velocity. Sensor nodes are specified by their location and a detection range. Time is incremented and as each target flies within the detection range of any sensor pair, its location is noted. When a target is in range of any node and any other target is in range of any other node and if two azimuth strobes intersect the ghost is noted. Tracking is assumed (a ghost track is the set of false intersections due to the same real target pair observed by the same sensor pair in time). For all tracks, the velocity, the acceleration, the apparent power that is needed to be generated at the intersection location given the observed power at each detecting sensor, and the rate of change of that apparent power are calculated.

Figures III-1 through -5 show these functions for the case of 2 targets and a single sensor pair. In Fig. III-1, T_1 and T_2 mark the initial target locations. Speed for both targets is 250 m/sec and radiated acoustic power is 120 dB. Locations are shown every 2 sec. S_1 and S_2 give the 2 sensor node locations. Ghost tracks begin at G_1 and G_2. Recalling that adjacent points in the ghost tracks are also at 2-sec intervals, the G_1 track is seen to slowly accelerate out of the S_1 node and to slow to near zero velocity as it approaches the S_2 node. The G_2 track flies in from large range at very high velocity and exits in the same manner. Figures III-2 and -3 show the calculated velocity and acceleration for each track in Fig. III-1. The true target tracks appear with fixed velocity and zero acceleration. Mach 1 and a nominal velocity for a very slow jet are also shown in Fig. III-2. Physical reasonableness of velocity and acceleration are clearly useful deghosting aids. Figure III-4 shows the apparent power attributed to each track by each sensor. Sound levels for various different types of jets are also shown. Each ghost not only exhibits wide power variations over its track but inconsistent values from sensor to sensor at each point in time. The time derivatives of the average track power for each track are shown in Fig. III-5.

Figures III-6 and -7 show the resulting tracks for 2 targets at 3 sensors and 3 targets at 2 sensors, respectively. The variations in the ghost tracks again point to unusual and, at times, impossible target behavior. With respect to using local redundancy as a deghosting parameter, Fig. III-8 shows one instant in time of the simulation in Fig. III-6. If the small square in the figure represents system location resolution then the two ghosts with it are indistinguishable from one another. Since these two ghosts are the result of three bearings, just as each target, there is as much confirming evidence for the ghost as for the targets.

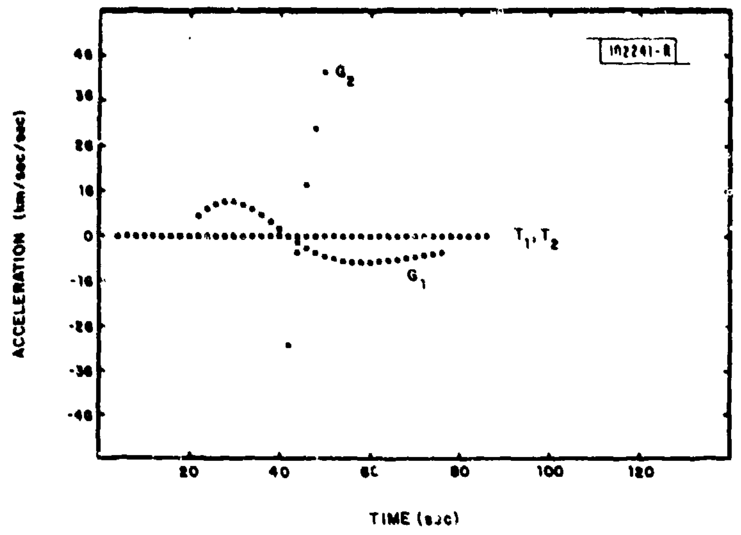

Fig. III-3. Accelerations of the ghosts in Fig. III-1.

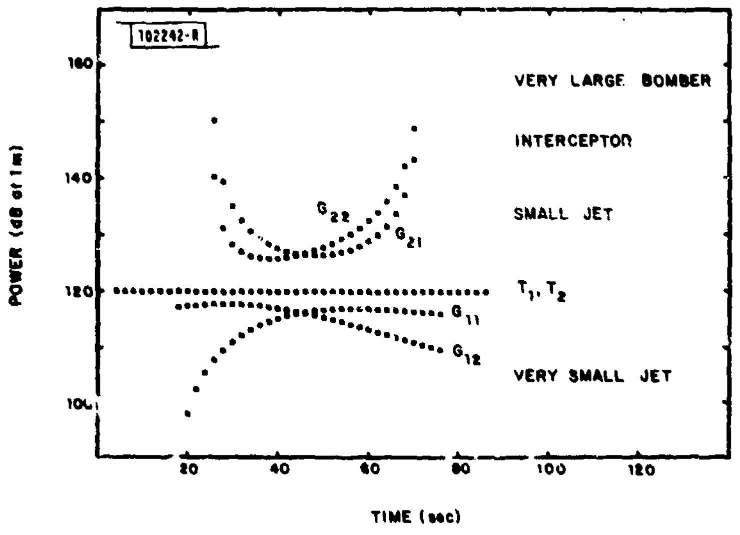

Fig. III-4. Power that would be estimated for a target at each ghost position in Fig. III-1 given the observed power at each sensor. G_{IJ} = apparent power of ghost I at sensor J.

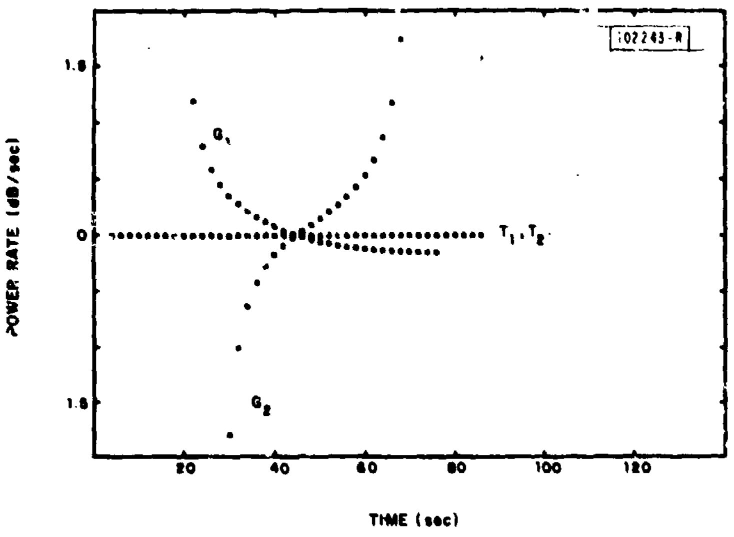

Fig. III-5. Power time rate of change that would be associated with the ghosts in Fig. III-1.

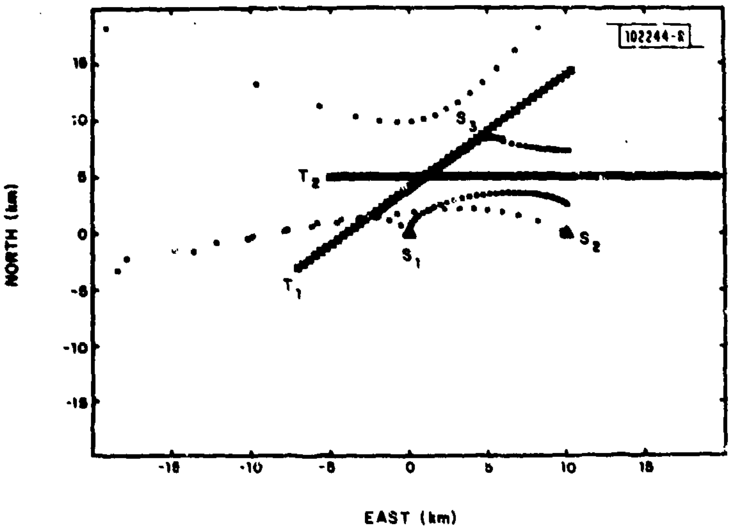

Fig. III-6. Same as Fig. III-1 with the addition of another sensor node S_3. Effect is the generation of 2 additional ghost set pairs.

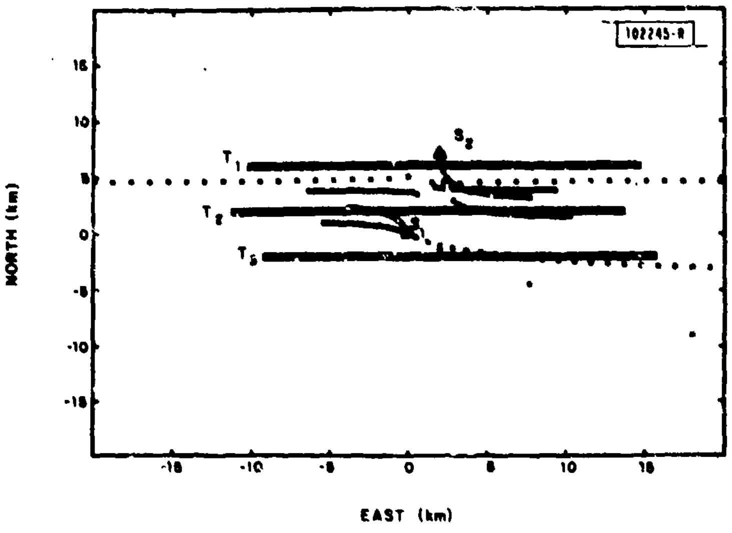

Fig.III-7. Ghosts associated with a single sensor node pair but for 3 targets.

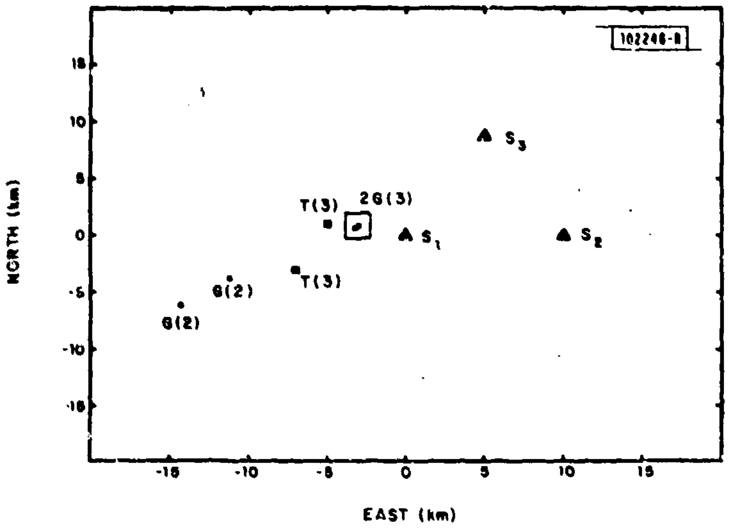

Fig.III-8. Snapshot of target and ghost locations at one instant of time. If the resolution of the sensors corresponds to a location error the size of the small box, the 2 ghosts within the box would be indistinguishable and their 3 associated azimuth strobes would produce the same degree of redundant information as would be attached to the actual targets.

The above simulations are generally characteristic of all the simulations we have conducted.

In summary, software has been written and used to examine the ghosting problem in a DSN. From a number of examples it was established that track dynamics in a DSN geometry are robust deghosting parameters. Additionally, we found that acoustic signature estimation, primarily perceived power level, could also be used to deghost. Local redundancy cannot be used exclusively for deghosting; however, network redundancy (i.e., ghost track consistency from sensor pair to pair) is a strong deghosting aid. Though we did not investigate the algorithms needed to do the deghosting nor did we account for the effect of acoustic propagation in determining track parameters, the work does indicate that the structure of a DSN designed for tracking low-flying aircraft by acoustics is such that the ghosting problem, though computationally demanding, is solvable.

IV. DATA ACQUISITION SYSTEM

Both the hardware and the software for our Data Acquisition System have been made operational. The following sections summarize hardware and software activities related to this system.

A. HARDWARE

The Data Acquisition System described in our previous SATS* is now operational. A nine-element array has been installed on the roof of one of the Laboratory buildings, and preliminary data have been acquired. Analysis of these data showed that there was significant system noise masking the data, and considerable effort has been expended in reducing that noise to an acceptable level. This section reports on the configuration that resulted in a satisfactory system noise level and also on how some of the other problems, such as providing overload protection for microphone preamplifiers, were solved.

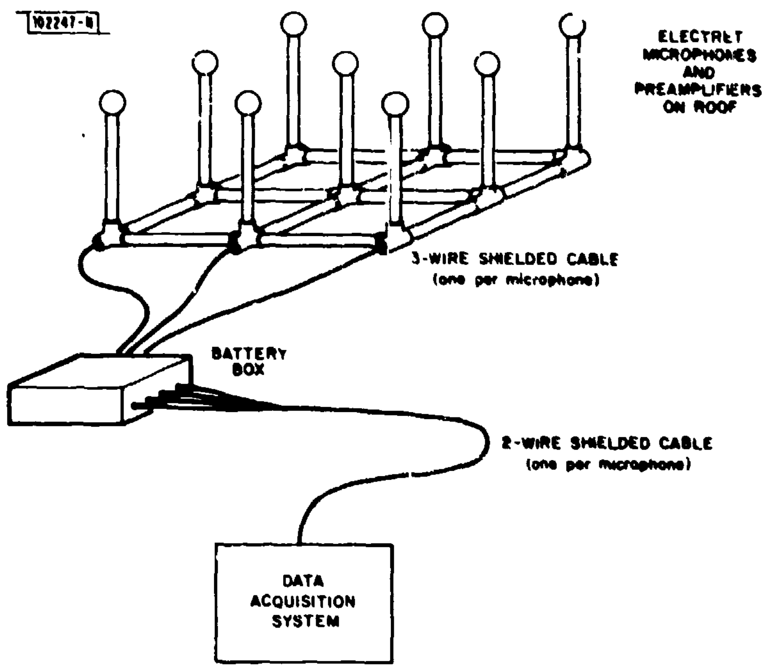
102247-N
ELECTRET MICROPHONES AND PREAMPLIFIERS ON ROOF
3-WIRE SHIELDED CABLE (one per microphone)
BATTERY BOX
2-WIRE SHIELDED CABLE (one per microphone)
DATA ACQUISITION SYSTEM

Fig. IV-1. Data collection system.

Figure IV-1 shows the configuration used for the microphones. The array is a 3 by 3 symmetric design with a 1-m spacing. This configuration is not necessarily optimum and we expect to experiment with others in the future. The particular configuration was mechanically easy to construct and the regular rectangular grid will allow use of computationally efficient array processing algorithms which exploit the regular grid.

*Distributed Sensor Networks Semiannual Technical Summary, Lincoln Laboratory, M.I.T. (30 September 1979), DDC AD-A086800.

PRECEDING PAGE BLANK-NOT FILMED

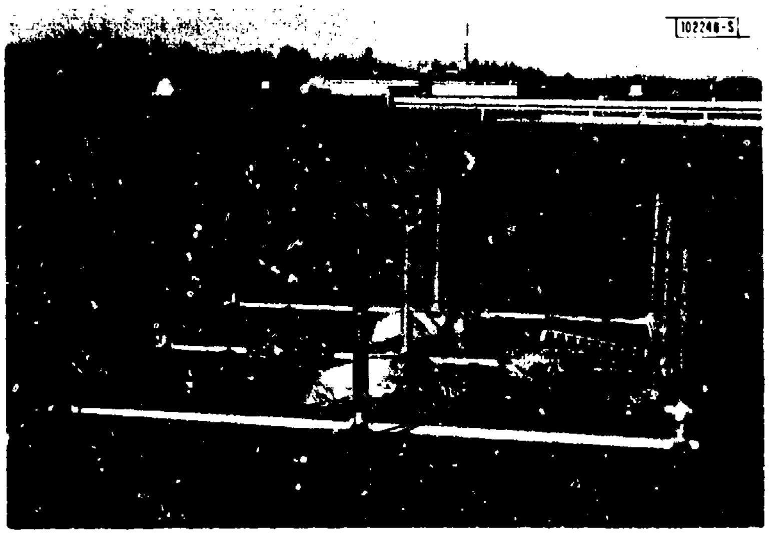

Fig. IV-2. Acoustic array on L-Building roof.

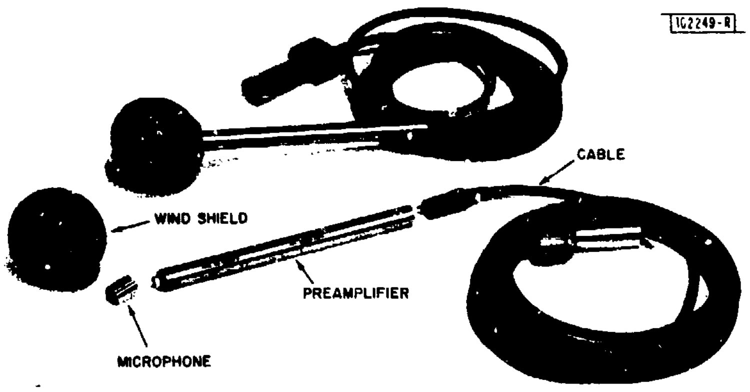

Fig. IV-3. Microphones.

The physical construction of the array is shown in Fig. IV-2. The support stand is made from 2-in. pipe, braced with threaded cross ties that enable the array to be squared. The microphones are supported by open-cell foam rubber supports in the vertical pipes. This minimizes problems with microphonics and enables final leveling of the microphones to a horizontal plane. The cables run through the pipes and then out to the battery box.

Figure IV-3 shows the microphone assemblies. The microphones are electret microphones mounted in a ported housing. These were chosen because the element is able to resist dampness, and the frequency response and sensitivity are adequate at frequencies as low as 10 Hz. The microphones are mounted on top of preamplifiers which convert the weak pressure-induced change of capacitance of the electret element into a substantial current capable of driving a 1000 ft or more of cable. The cable to the battery box then plugs into the base of the preamplifier, making for an assembly that is weatherproof. The microphone itself is surrounded by a 2.25-in. open-cell foam wind shield.

Each of the microphones is connected to the battery box, by a three-wire cable as shown in Fig. IV-4. The output of the preamplifier is a balanced feed with a separate shield and power feed. The shield is connected to the cable shield and the casing of the battery box. The output from the battery box is a two-conductor shielded cable for each microphone. Here again, balanced feed is used to minimize noise.

Within the battery box, three mercury batteries are wired in series to provide the 20 V needed by each microphone. Separate sets of batteries are used for each microphone to minimize crosstalk in the power supplies. Each power line has an overload protection relay in series with it. This relay opens the circuit in the event that the microphone input is shorted such that the preamplifier draws more than 15 mA. Once off, the power is latched off until the reset button is pressed for that relay.

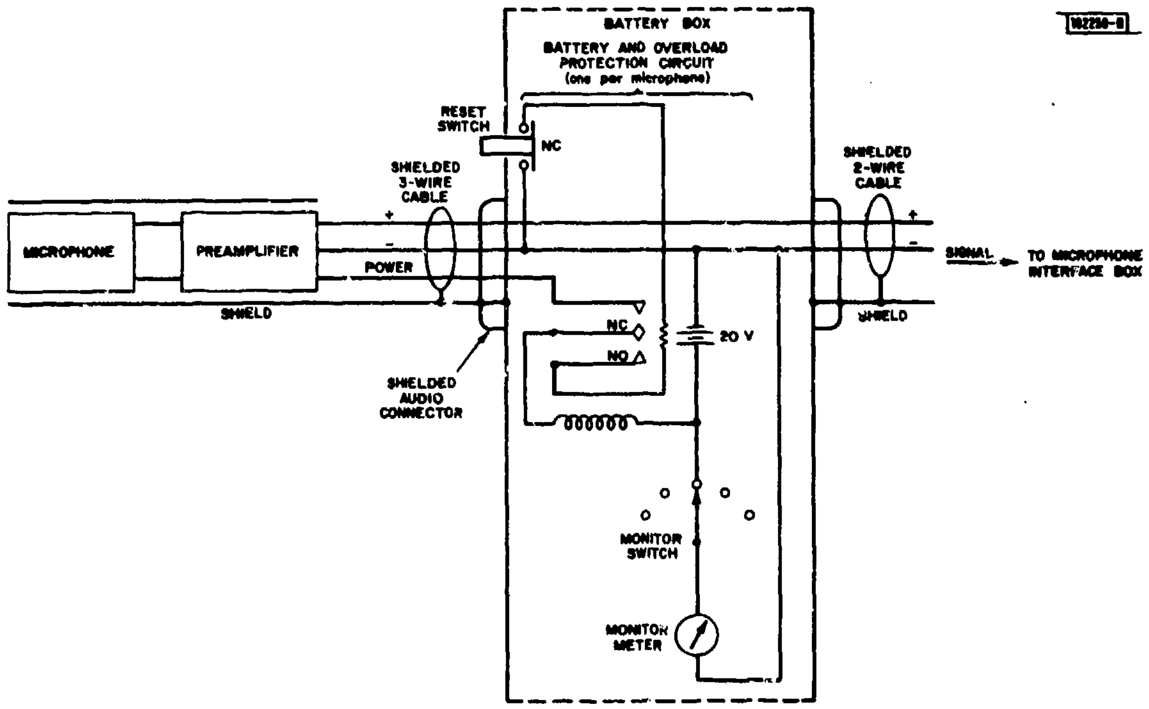

Fig. IV-4. Shielding and power supply arrangement for microphones and preamplifiers.

Correct interconnection of the signals, the analog grounds, the shields, and the electrical supply neutral is essential for minimization of system noise. The principles that we followed were to use balanced feeds where possible, to avoid ground loops, and to ensure that the analog ground only connected with the power neutral at one point.

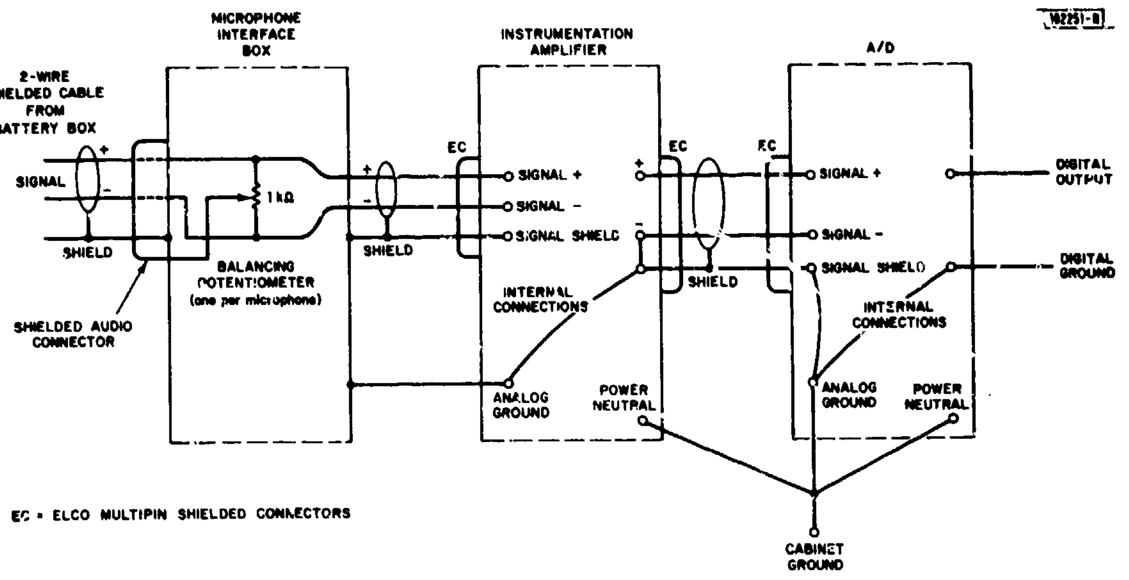

Fig. IV-5. Shielding and balancing arrangement for instrumentation amplifier and the A/D.

As shown in Fig. IV-5, a 1-kΩ potentiometer was used to balance the signals with respect to the line. This 1-kΩ resistor serves to discharge any charge built up between the signal lines and the shield. Without this resistor there is no discharge path and very high voltages can arise which exceed the instrumentation amplifier common mode rejection limit of 5 V. This potentiometer is used to balance the noise components picked up on the line.

The instrumentation amplifier has a shielded balanced input, but only a single-ended output with the signal minus connected to analog ground. The input to the A/D is balanced with respect to its analog ground. However, we had to feed this single ended because of the single-ended output of the instrumentation amplifier. Ground loops and noise were minimized by connecting signal minus to signal minus between these units with local connection to analog ground on the A/D and no direct connection between analog grounds except through the signal cables.

For safety reasons all units must have their shields connected to the neutral of the electrical power supply at some point. By making this a single point connection, noise was minimized. To this end, the microphone connector box is isolated from its front panel in the rack, so that power noise is not induced in the shield at this point. The box and the microphone plugs have been installed in the middle of the rack so that noise cannot be introduced by human contact.

When these precautions are taken, the instrumentation amplifier noise is typically 150 μV in a 10-Hz band when measured with a dummy microphone on the preamplifier. This represents a level that corresponds to a 20-dB level of sound at the microphone. To put this in perspective, we note that ambient conditions where the microphones are now located are typically 50 to 60 dB and even very quiet locations and conditions would be no less than 30 to 40 dB. Usually the peak level of system noise is at 60 Hz, although we have seen cases where RF breakthrough of the anti-aliasing filters in the instrumentation amplifier is significant. The overall system is set so the gain-ranged A/D conversion system will operate from the noise floor up to 114 dB, which is large enough to handle very large aircraft at very close range without saturation.

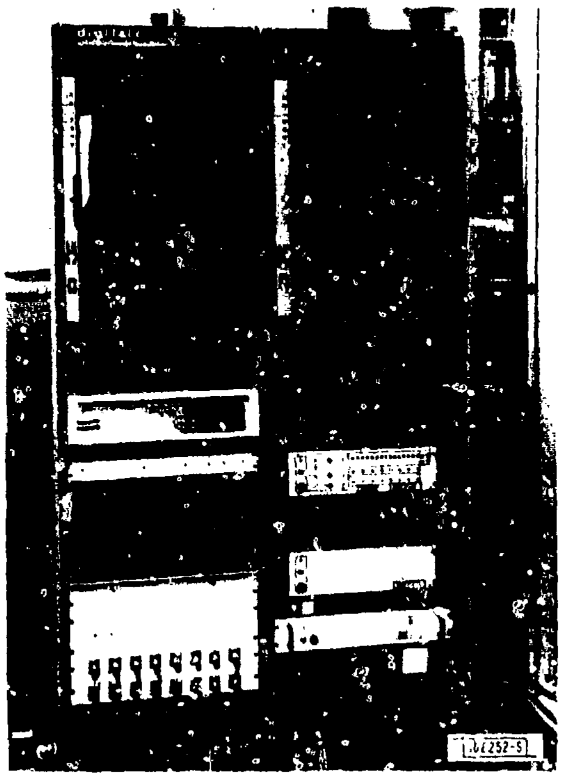

Fig. IV-6. Data Acquisition System hardware.

The configuration of the Data Acquisition System during the preliminary tests is shown in Fig. IV-6. This has now been modified to isolate the microphone inputs which are shown at lower left. When this system is converted to a DSN node, a third rack will be added to hold the array processor. Based upon experience with the acoustic array, battery box, and cable configurations described above we are now in the process of engineering the actual configuration which will be incorporated in other DSN nodes. Changes are being made primarily to further weatherproof the system and make it easier to routinely operate and maintain.

B. SOFTWARE

In our last SATS (September 1979) we described the Data Acquisition System (DAS) program and the Data Acquisition Kernel (DAK) that supports DAS. These programs have now become operational. DAS consists of 1,969 lines of code and 483 lines of short-form documentation, while DAK consists of 10,334 lines of code and 2,119 lines of short-form documentation. (Short-form documentation is compact documentation suitable for reference use.)

The following four DAS modules are defined:

usr	The user interface.
rec	The recording server, a foreground process that does the actual data copying and also delivers sample data to the user interface for display.
dar	The data record module, which formats data for magtape I/O.
par	The parameter record module, which formats run parameters for magtape I/O.

The rec and dar modules are finished, and DAS currently consists of these modules plus a small test program that serves as a user interface. The current simplified user interface, though not well suited to naive users, is functionally adequate and no substantial enhancements are currently scheduled. Implementation of the par module defining parameter records has been held up by memory space limitations of the current version of DAK. Sufficient space will be available under the next version planned for release in July. We will then convert DAS to the enhanced version of DAK and add parameter record capabilities.

As noted above, the major current limitation on use of the DAK is that all programming, including DAK and user code, must fit into 57K bytes of memory, the maximum size of a [illegible]tual address space on a 16-bit minicomputer such as a PDP-11, minus 8K bytes needed for the PDP-11 I/O register page. We are now enhancing DAK to allow the user to address 262K bytes of memory indirectly through a system call that copies any contiguous part of that memory to any other part of the memory. I/O devices will also be able to address the full 262K bytes of memory, allowing any part of that memory to serve as an I/O buffer. Thus in effect, the user will have 262K bytes of memory, only the first 57K bytes of which can be addressed by normal code. This enhanced version of DAK will be used to support signal processing as well as data acquisition functions in initial three-node experiments.

This system for adding to the memory available to a 16-bit minicomputer user process is simple and suitable for interim use. However, over the next year we plan to replace DAK with a more substantial revision called OSD/RENE, which will support multiple virtual memory spaces on one computer as well as distributed programming using multiple computers. OSD, the Object Structured Discipline, is a revised version of the basic software layer of DAK, and is usable for computation, memory management, and multiprocess scheduling within a single virtual address space under any operating system. RENE, the Real-Time Network Kernel, will be built on top of OSD and will provide fundamental communications servers and multiple memory space management servers. One goal of OSD/RENE development is to provide a version which runs underneath UNIX, as well as in an independent PDP-11, thereby allowing software to be shared between the real-time node computers and a UNIX-based analysis and program development computer. See Sec. VI for more details on RENE design.

V. SIGNAL PROCESSING

Work has continued on non-real-time signal-processing software for use in conjunction with algorithm development and work has started on real-time software which will make use of special real-time hardware. The following two sections report on these task areas.

A. ACOUSTIC ANALYSIS PROGRAM

One of the major technical activities in the area of signal analysis has been the development of the Acoustical Data Analysis Program (ADAP) for our general research computer. The function of this computer program is to take data from the Data Acquisition System and analyze it, producing output in a form suitable for input to tracking algorithms. This section describes ADAP and some of its capabilities. The computer which supports ADAP is our PDP-11/70 running the UNIX operating system.

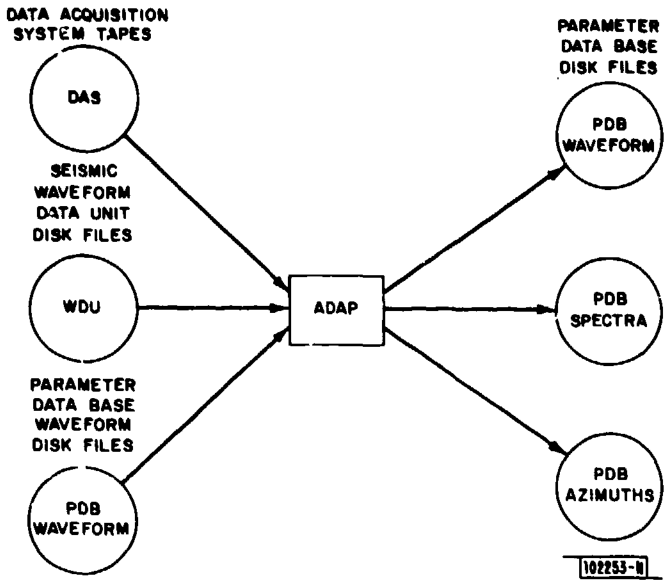
DATA ACQUISITION SYSTEM TAPES
DAS
SEISMIC WAVEFORM DATA UNIT DISK FILES
WDU
PARAMETER DATA BASE WAVEFORM DISK FILES
PDB WAVEFORM
ADAP
PARAMETER DATA BASE DISK FILES
PDB WAVEFORM
PDB SPECTRA
PDB AZIMUTHS
102253-N

Fig. V-1. Input/output diagram for ADAP.

As shown in Fig. V-1, ADAP can accept data from DAS tapes, from seismic Waveform Data Unit (WDU) disk files, and from Parameter Data Base (PDB) waveform disk files. Its output is one of three types of PDB files, namely waveform, spectra, and azimuthal power files. In creating these files ADAP converts the input data into fixed blocks, performs Fourier transforms to produce the spectra, and performs maximum likelihood method (MLM) or conventional beamforming to produce the azimuthal data. ADAP also has the capability to produce plots of the waveforms, spectra, and the azimuthal data.

The DAS tapes contain 4K byte blocks of data in gain-ranged 16-bit word format. ADAP demultiplexes the data and decodes their format prior to any analysis. Because DAS tapes are collected in real-time there may be tape errors which could not be corrected. Thus, ADAP checks for proper time sequence of records, makes other error checks, and selects out sequences of good data blocks for analysis. ADAP will accept input of test conditions such as array configuration and calibration information and, in addition to using this information internally, will include it in output files.

When the Ft. Huachuca data* were originally converted from analog to digital form, they were converted into our standard seismic data format WDU files. To enable ADAP to handle old data as well as new, it was given the ability to input this format. This also enabled us to use existing synthetic data generation programs to create test data for ADAP.

Parameter Data Base files have data in fixed-length records with keyed access based upon parameter values. The structure of the three types of PDB files is shown in Fig. V-2. Each PDB file has a header which contains access information followed by a variable block of text.

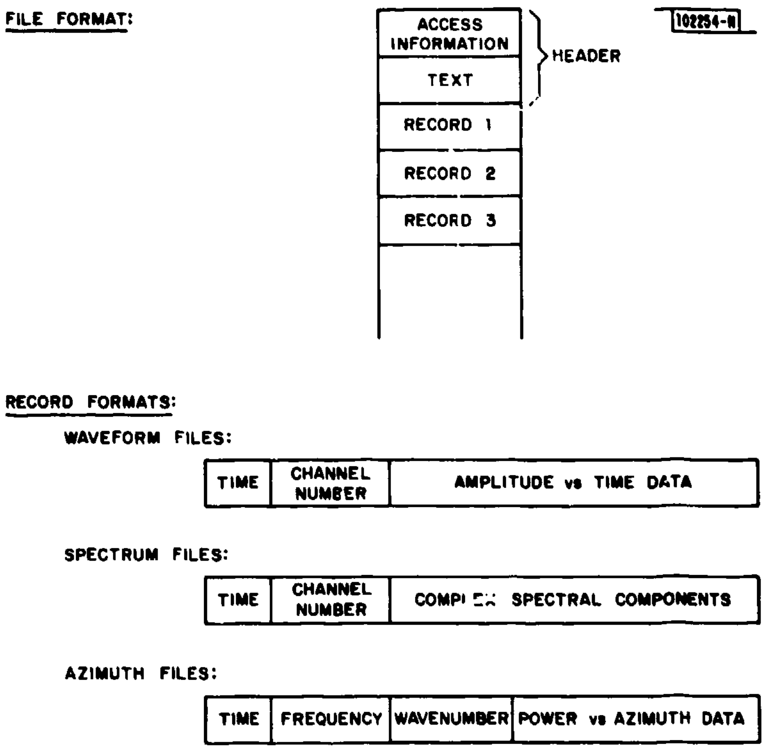

Fig. V-2. File formats for ADAP.

This text information contains a description of all the parameters pertinent to the data, such as the array configuration. The header is followed by fixed-length records which contain one or more access parameter values followed by a data block for those parameters. The simplest PDB file format is that for waveforms. Here the amplitude data for one channel for a given number of samples are written into a record. The time key is the time of the first data point.

*Distributed Sensor Networks Semiannual Technical Summary, Lincoln Laboratory, M.I.T. (30 September 1977), DDC AD-A050160.

Spectral files are similar except that each record contains the Fourier transformed data from a waveform data block.

Azimuthal files have records containing power vs azimuth, parameterized by time, frequency, and normalized wavenumber. In tracking we need to find peaks at each time as a function of frequency, azimuth, and wavenumber. This involves searching through the azimuthal data, which is made very easy by the direct access mechanism provided by the PDB subroutine package which was developed under another program for seismic use. It was for this reason that the PDB file structure was chosen. The spectral and waveform PDB files were added for compatibility and convenience.

In performing spectral analysis, the user can select window functions. In computing power vs azimuth, the user can select from conventional and MLM adaptive beamforming. The covariance matrices can be normalized if so required, and they can be stabilized by adding fractional noise along the diagonal. The user can select the block length, the spectral analysis parameters, and the number of blocks over which to average in forming the covariance matrices. The user can also select the number of azimuths and the normalized wavenumbers at which to perform the analysis.

Techniques of frequency selection for performing spatial (power vs azimuth) analyses are currently under development. Prior to the analysis of each set of blocks over which we are going to average, we must select the frequencies at which we are going to do the analysis. The process of determining power as a function of azimuth is computationally intensive. It is therefore desirable to restrict the number of frequencies to those which will yield data of interest.

The current method of frequency selection is to select frequencies which had the highest power during the previous analysis interval. (Analysis intervals are typically 1 sec long and separated from each other by 1 sec.) The spectrum used for frequency selection is the average spectrum over all the channels. We have also started to investigate the use of maximum entropy spectral analysis for the purpose of frequency selection and power spectral density estimation. It is expected that frequency selection devices will eventually be added, such as observing frequencies at which we have already acquired a target. Also, there exists the possibility of picking frequencies to maximize spectral discrimination.

The plotting capabilities of ADAP are currently limited to plotting a single record of a PDB file. The plotting capabilities are now being expanded to allow multiple channels for an arbitrary time period, aggregate spectra for multiple channels, and azimuthal data for multiple time periods to be displayed on the same plot. ADAP is also being expanded to allow the creation of WDU files from DAS tapes or PDB waveform files. This is being done so that tools already developed for analyzing seismic data can be applied to our acoustic data where appropriate.

B. REAL-TIME SIGNAL-PROCESSING SOFTWARE

The DSN test-bed node design contains a DEC PDP-11/34 minicomputer and a Floating Point Systems AP-120B array processor (AP). The AP will perform the signal processing for detection and tracking of targets. The AP is a single instruction stream, multiple data path, pipelined processor capable of performing one floating point addition or multiplication every 167 nsec. The AP's multiple data paths permit paralleling of computations to increase throughput. The PDP and the AP are linked by the PDP-11 UNIBUS which can be used for either programmed input/output or DMA transfers. The following summarizes current design approaches for AP software. This software will be integrated with DAK for interim use and with OSD/RENE for longer term use. (See Secs. IV-B and VI for DAK and OSD/RENE details.)

Communication between the two processors through the UNIBUS will be under the control of an "analysis server." This software is a module that will reside in the 11/34. It has three principal subdivisions corresponding to the functions it performs. It will implement a virtual array processor to accept high-level commands from user processes in the 11/34; it will contain a manager for AP memory; and it will incorporate a driver to execute commands on the AP hardware. The analysis server will enable the DSN user to execute signal-processing algorithms on the AP while minimizing the user's programming task. The user's communication with the analysis server is similiar to that with other DAK servers. Requests for computation will be sent to the server and results returned via queueable objects. In its role as manager of AP memory, the analysis server will maintain maps of AP data and program memory to avoid conflicts between programs. Finally, the server will have low-level driver software to read/write AP data and program memory, read/write internal and device registers, initiate DMA and programmed data transfers, and monitor interrupts.

A typical DSN user process will execute algorithms described elsewhere in this and previous DSN SATS. During this year, the principal requirement will be to execute frequency-domain beamforming algorithms in real-time on data collected from the DSN acoustic arrays. To perform a calculation, the user will send a request for service to the analysis server's queue. The requests are objects in the sense of OSD. A request object – called an an_op (analysis operation) will consist of a control header similar to an io_op (an input/output operation in DAK), a variety of parameters necessary to control execution of the function, the name of the function (e.g., FFT, matrix inversion, power as a function of azimuth and elevation, etc.), and the location of the input data on which the function is to be calculated. For example, a typical user sequence might be:

Code	Effect
an_oalloc(op)	Creation: Storage is allocated for the analysis operation, which the user calls "op."
an_oinit(op, parameters)	Initialization: The user specifies parameters which are not expected to change between calculations. These include the name of the analysis operation, the function, the operation priority, the analysis server (in the case more than one AP is available to the user), the queue to which to return the operation after processing, etc.
an_oset(op, buffer, size)	Setup: Data for a particular calculation are specified.
an_power(op, elev, freq, azimuth)	Execution: The operation is queued on the analysis server queue and the analysis server processes the operation. In this case, a high-level request is being made to calculate acoustic power for elevations, azimuths, and frequencies as specified by input objects "elev," "freq," and "azimuth."

The analysis server will process its queue according to the following algorithm:

(1) If there are no operations on the queue, wait until one arrives.
(2) Mark the first operation, op, on the queue BUSY.
(3) Interpret parameter fields of op.
(4) Download op to AP.
(5) Wait for op to return.
(6) Mark op DONE or ERROR as appropriate.
(7) Dequeue op.
(8) Return op to user's queue.
(9) Go to 1.

When the analysis operation object is returned to the user, it contains the function output, if the computation was error-free, otherwise status information in the control header indicates the type of exception encountered during the computation. The user has the freedom to respond to exceptions in the manner appropriate to his application.

The analysis server will manage all the AP's resources. The server will contain memory allocation maps for data and program memory. The AP main data memory is to be partitioned into two parts: a static region of user allocatable memory, and a work region. In the static area, the user may store results for input to other algorithms. For example, the user might allocate accumulators for cross-correlation matrices, power averages, etc. Memory in the work space is to be automatically allocated for each routine loaded in program memory. Depending upon the type of computations being performed, it is possible that not all the user's programs can be loaded in AP program memory at the time of system start-up. In this case, routines will need to be downloaded when required. The loading will be managed by the server.

The current design calls for a fairly conventional software driver to perform the low-level interaction with the AP hardware. It will be constructed from a library of routines for reading and writing internal AP registers and AP device registers. Work on these routines has already begun in order to perform acceptance tests on an FPS AP-120B purchased by another group in Lincoln Laboratory. The operation of the driver will consist of responding correctly to AP interrupts. When an AP interrupt occurs, the driver must decide whether it indicates completion of an analysis operation or an error condition, and respond accordingly. If an operation has been completed successfully, output results will be stored. A new operation, if present on the driver queue, will be loaded. Data will be transferred by DMA under AP control. The driver will then start the AP executing the next program wait until an interrupt occurs again. This cycle repeats itself indefinitely.

VI. REAL-TIME NETWORK KERNEL DESIGN

In the previous SATS[1] we defined a real-time network kernel as an operating system containing network communications, process scheduling, basic memory management, and no I/O drivers. In such a system the I/O drivers and user processes are coequal users of the kernel and they communicate with each other exclusively through the kernel network communications system. The logical network communication mechanisms must be good enough to support all communications, including those between real-time I/O drivers and real-time user processes. We further require that communicating processes should be made insensitive as to whether or not they reside in the same virtual address space, in different spaces on the same computer, or on different computers. The only difference in these circumstances should be differences in throughput and turnaround time.

Queueable objects are the mechanism we plan to use as the basis for interprocess communication. Based upon this idea we have developed a real-time kernel, the Data Acquisition Kernel, that we feel presents an approximately correct user interface for a real-time network kernel, even though DAK itself requires all processes using it to reside in the same virtual address space. DAK is now debugged and in use (see Sec. IV-B). While we were implementing DAK, we studied the problems that would arise in building a true multicomputer real-time network kernel, and identified first cut solutions. It is these problems and solutions that are discussed below. We are now in the process of carrying out the detailed design and initial coding of a successor to DAK, called RENE (REal-time NEtwork kernel), which embodies these solutions and which will be our first version of a truly multicomputer system. In the meantime, we shall continue to use DAK, which is very similar in structure to RENE.

A. QUEUEABLE OBJECT STRUCTURE

A queueable object is a structure beginning with a special queueable object header. DAK queueable object headers initially contained a few list pointers that allowed the objects to be placed on queues. The prototype queueable object was the input-output operation, or ioop. We found that many ioop structure elements were needed in virtually all queueable objects, and began the process of moving these to the queueable object header. This process is still continuing: in RENE we expect the header to contain an 8-bit command code, 32 option flag bits, an 8-bit termination code, 32 status flag bits, a pointer to the object's user queue (see below), an optional pointer to a character string print name for the object, a 32-bit sequence number, and a circuit designation (see below) that behaves like a priority-level specification. In addition, there are elements specialized for transmitting the object through a network. These record when the object was last sent, whether and when its receipt was acknowledged, whether and when retransmission has been requested, and the formats governing transmission and reception of the object.

In DAK, running on a PDP-11/34, the CPU overhead time required to transmit a queueable object is on the order of 100 μsec, and the consequent limit on the total system object transmission rate is several thousand objects per second. But, when processes do not share an address space, transmission overhead per object increases substantially, so that we expect a typical minicomputer to be able to transmit or receive only several hundred objects per second either between its own separate virtual address spaces, or between itself and the outside world. Consequently, it will be necessary to transmit mostly large objects between separate address

PRECEDING PAGE BLANK-NOT FILMED

spaces or between a computer and the external world. We will retain the option of improving communications efficiency by putting two processes in the same virtual address space.

B. QUEUE ORGANIZATION

When communicating processes are in the same address space, as in DAK, one process (called the server) owns a queue, and the other process (called the user) sends an object to the server by enqueueing it on the server's queue. The server processes the object, and passes it back to the user by enqueueing it on a return queue owned by the user. This is the standard method for communicating with queueable objects, as described in Sec. IV-B-2 of the previous SATS.[1] We will retain this system in RENE when the user and server processes are in the same virtual address space.

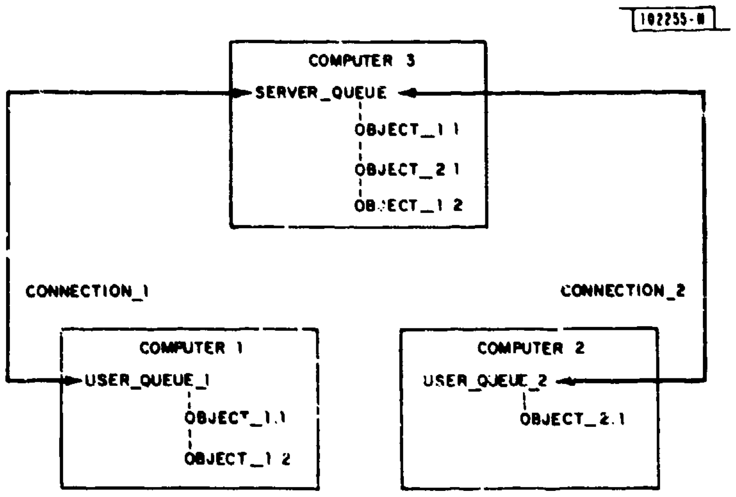

Fig. VI-1. Queues, connections, and objects in RENE.

If the user process is in a different address space from the server process, an extra queue, which we call a user queue, must be introduced (see Fig. VI-1). The user queue resides in the user process address space and holds a copy of all objects sent by the user process to the server queue in the server process separate address space. The user queue is a surrogate for the server queue: it represents the server queue to the user process. When the user places an object on the user queue, the object is copied onto the server queue, where the server processes it, and copies the object back to the user queue, from whence it may be dequeued and returned to a separate user process return queue. While the server processes the object there are in fact two copies of the object: one on the user queue and one on the server queue. Only object elements read by the server need be sent from the user to the server, and only elements written by the server need be returned to the user; so that transmission formats determining which elements to transmit need to be provided for each object type and direction of transmission.

The user and server queues are coupled by a connection. One server queue can be connected to many user queues, but each user queue can be connected to only one server queue. Each of the many user queues connected to the same server queue represents only part of that server queue, and contains only objects sent to that server queue by user processes in the same address space as the user queue. In addition, the server queue may contain objects sent by processes in the same address space as the server process. For such objects there is no separate user queue.

There are a number of advantages to such a scheme. One advantage has already been mentioned: the use of a single communications interface that works with improved efficiency when processes share an address space but also works with normal efficiency when the processes are in different spaces. Two other advantages are that an initial copy of each object is always available in the user process for retransmission, and that the user process provides all its own buffers in a direct and simple way that gives it control of communications buffering and can relieve the network and server of difficult buffering problems.

C. CONNECTION MOBILITY, RECONNECTION, AND PROTECTION

In DAK we found it convenient to pass servers from one process to another. For example, a user interface process might open a tape drive server, write labels on the tape, and then pass the whole server to a real-time process that would record data on the tape. When the data recording was done, the recording process returned the tape drive server to the user interface process.

In a network or multiple address space environment a user queue, which may be passed among processes, is an analog to a DAK server which can be passed around between processes. To implement such user queue passing we propose to use two levels of connection: real and virtual. A real connection is between a user queue in a particular address space and a server queue in another particular address space. A virtual connection is a sequence of real connections all having a common user queue object, though that object may migrate from address space to address space without breaking the virtual connection. Processes using normal system calls see only virtual connections.

At any time the real connection that implements a virtual connection may become broken. For example, a real connection will be broken when its user queue is passed between address spaces. It is also possible for a connection's server queue to disappear, and the virtual connection must then make a real connection with a different server queue. This could be the result of a computer crash or could be scheduled by the server, for example, to enforce adherence to new protection or access rules it has adopted.

Our approach for handling these situations is to endow every virtual connection with a pathname and status information that can be used to remake the real connection whenever necessary. This pathname is generated when the virtual connection is first established. The pathname does not define a route through the network but unambiguously identifies the process or file to which a connection is made. In the case of connections to a file, the pathname identifies a specific version so that reconnection can be unambiguously made to the correct version or cannot be made at all if the version no longer exists. Pathnames may be sent to directory servers, which may either set up a server to be the other end of the real connection, or name another directory server to which the pathname should be sent. Servers for virtual connections will also be identified by pathnames.

D. RELIABILITY

One approach to distributed system reliability is to require that each user process be made responsible for its own reliability, and be able to survive such network and server failures as commonly occur. This is the approach we are taking and it is a major factor behind our communication design.

Server reliability is one very important problem. Experience with existing network servers and with modern hardware I/O controllers, which are analogous to software servers in many ways, has indicated that it may not be possible or practical to write servers that correctly recover from all error situations. Therefore, we propose to defend against server failures by requiring servers to be reinitializable, where possible. A reinitializable server is constructed so that the user retains a precise copy of server status, at least as it affects the user, and if the server fails, the user simply reinitializes the server, supplying the proper status to return the server to the state where it left off. In this way the user is able to recover from a server failure. An example is an open disk file I/O server, with the user process retaining the pathname of the file, the read-write access privileges for the open file, and the current position within the open file.

A second major reliability problem is loss of messages by the network. The only cure for such network failures appears to be time-out driven retries. Moreover, experience with existing networks has led us to the viewpoint that if the user process is to be reliable, that process must be the process which uses time-out driven retries and we intend to design and implement software adhering to this point of view.

To support user driven retries and still defend against server memory overload we introduce the concept of a retryable request. A request is an object sent by a user to a server, which processes the object and returns results as part of the object. A retryable request has the property that if the same object is transmitted several times to the server, and if the server fully processes at least one copy of the object, then the effect on the server will be the same as if the object was sent only once and was fully processed. Furthermore, if several object copies are fully processed by the server and cause several copies of results to be sent back to the user, then there must be some way of constructing a single final result which is the same as if a single copy of the request was processed by the server and its single complete result received by the user. Reliability of retryable requests is in the hands of the user process. Use of the retryable requests by servers avoids the need for the server to keep copies of objects being returned to users until the server receives acknowledgment from the user that the returned object has been received. An example of a server which can be designed to use retryable requests for the bulk of its activities is a random access memory server, such as a disk file manager, or a virtual address space memory manager; for an example see the Xerox PARC Distributed File System.[2] Of course, some nonretryable requests will be unavoidable and we expect to handle them on a negotiated basis with the server accepting the request when there is no danger of server memory overload.

Note that although users control time-out driven retries, the network and servers also can and should request retransmission in many circumstances. For example, when confronted with congestion the network may discard objects in its transmission buffers, and request the user to retransmit these after a delay (for example see a proposed congestion control algorithm for AUTODIN II in Sevcik).[3] In another example, if a server receives an object completely, and some time later is forced to discard the object in order to free memory space for higher priority requests that arrived after the object, then the server may ask the user to retransmit the object at an appropriate time.

Another reliability-throughput problem arises when a user sends many requests at once to a server, the first request is temporarily lost, and the other requests must wait for the first. To combat this we propose that servers be allowed to process requests in non-sequential order where possible.

REFERENCES

1. Distributed Sensor Networks Semiannual Technical Summary, Lincoln Laboratory, M.I.T. (30 September 1979), DDC AD-A086800.
2. J. E. Israel, J. G. Mitchell, and H. E. Sturgis, "Separating Data From Function in a Distributed File System," Proc. 2nd International Symposium on Operating Systems: Theory and Practice (Elsevier North-Holland, New York, 1979), pp. 17-27.
3. F. J. Sevcik and P. J. Nichols, "A Flow Control Technique for Datagram Subnetworks," National Telecommunications Conference (IEEE Press, November 1979), pp. 32.21-6.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

19 **REPORT DOCUMENTATION PAGE**	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: 18 ESD-TR-80-73 / 2. GOVT ACCESSION NO.: AD-A092 766	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (*and Subtitle*): 6 Distributed Sensor Networks.	5. TYPE OF REPORT & PERIOD COVERED: 9 Semiannual Technical Summary rept. 1 Oc[illegible]979 – 31 Mar[illegible]80
	~~6. PERFORMING ORG. REPORT NUMBER~~
7. AUTHOR(s): 10 Richard T. Lacoss 12 42	8. CONTRACT OR GRANT NUMBER(s): 15 F19628-80-C-0002, ARPA Order-3345
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Lincoln Laboratory, M.I.T., P.O. Box 73, Lexington, MA 02173 207650	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: ARPA Order 3345, Program Element Nos.61101E and 62708E, Project Nos.0D30 and 0T10
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Advanced Research Projects Agency, 1400 Wilson Boulevard, Arlington, VA 22209	12. REPORT DATE: 11 31 Mar[illegible]80
	~~13. NUMBER OF PAGES~~ 44
14. MONITORING AGENCY NAME & ADDRESS (*if different from Controlling Office*): Electronic Systems Division, Hanscom AFB, Bedford, MA 01731	15. SECURITY CLASS. (*of this report*): Unclassified
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (*of this Report*)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (*of the abstract entered in Block 20, if different from Report*)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (*Continue on reverse side if necessary and identify by block number*)

multiple-sensor surveillance system
multisite detection
target surveillance and tracking
2-dimensional search-space cell
acoustic, seismic, radar sensors
low-flying aircraft
acoustic array processing
high-resolution search-algorithms

20. ABSTRACT (*Continue on reverse side if necessary and identify by block number*)

Distributed Sensor Network research in the areas of tracking, signal processing, and system software is reported as is progress with the design and development of a DSN test bed. Two-node acoustic location algorithms have been reformulated to better fit into the real-time DSN environment. Deghosting techniques have been investigated and the general organization of tracking tasks further refined. The development of hardware and software for a small acoustic array and data acquisition system which will evolve into a node of a DSN test bed has been completed and the data acquisition system is operational. System software designs and ideas for initial three-node test bed use and for more general DSN systems are presented.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

207650 xlt